



# ERNE Client v.1.0.0

**for the Early Detection Research Network**

# Table of Contents

# 1.1 Introduction

## ERNE Web Services

The Early Detection Research Network (ERNE) Resource Network Exchange (ERNE) is a distributed, network-based, web accessible series of services that enable lookup, description, profiling, and retrieval of specimen data. It's based on the Object Oriented Data Technology 's data grid services . It provides several interfaces to scientists, data analysts, and developers as *web services* using standard Hyper Text Transfer Protocol (HTTP) and Extensible Markup Language (XML) standards.

By providing interfaces using HTTP and XML standards, you can develop applications that use ERNE metadata and data from all modern software environments.

## Web Services

ERNE provides two web services:

- `https://ginger.fhcrc.org/prod`

  This service provides for retrieval of specimen data products over HTTP. You pass in parameters that tell what ERNE site to query and what specimen data to retrieve. In response, you get ERNE specimen products.

- `https://ginger.fhcrc.org/q`

  This service provides for resource location and description over HTTP. You pass in parameters that tells what kind of metadata to search and a search expression. You get metadata descriptions (called "profiles") back in XML format.

Both web services are centrally located, as you can see from the URLs, on a system at the Fred Hutchinson Cancer Research Center. Back-end software (the data grid framework) automatically directs your query throughout ERNE sites.

## ERNE Software

The data grid framework that the ERNE uses to implement its data systems are all Java-based. As a result, you can take advantage of Java software, included in this package, to simplify some tasks (specifically, profile retrieval). You aren't required to use Java, of course, but several "jar" files are included should you wish to.

The jar files included are:

- `edm-commons`. This jar contains common components that are needed by all the other jar files. You can always grab the latest version from the Common Components web site.
- `grid-profile`. This jar contains classes for the representation of profiles, which are metadata descriptions of resources. Using these classes means you don't have to work with the XML representation of profiles that come from the `https://ginger.fhcrc.org/q` web service. You can always grab the latest version from the Profile Service web site .

See the guide section on querying ERNE profile servers for instructions on using these two jars. (No jars are provided for product retrieval since Java includes everything necessary for handling products already. See the guide section on getting products for more details.)

# 1.2 Retrieving Products

## Getting Products

The EDRN Resource Network Exchange (ERNE) uses the OODT framework to transfer specimen data (products) between the various ERNE sites spread throughout the country and the central node at the DMCC. You can access products using any web browser or any program that can speak the Hyper Text Transfer Protocol (HTTP). That means you can also embed product retrieval into your own analyis applications or other programs, since virtually every programming language supports HTTP.

This section details what's involved in using HTTP to get specimen data (products) in ERNE.

## Product Servers

Each of the participating ERNE sites runs a *product server* that the central node at the DMCC uses to transfer product data. Product servers have the responsibility of converting queries from generic Common Data Element (CDE) based expressions to site-specific queries, as well as returning site-specific results as CDE-based results, all automatically.

To retrieve specimen data (products) via HTTP, you need to know two things:

1. The *object name* of the product server to which you'd like the query directed.
2. The *query expression* that selects and constrains what specimen details to retrieve from the distant product server.

When you present an HTTP query for a product, the central node at the DMCC passes that query onto your selected product server, gathers the results, and then streams it back to you over the HTTP conenction.

### Object Names

The table below lists each participating site and the *object name* of the product server that runs there. (Please note that as additional sites add product servers this list will be out of date. Check with the DMCC for the latest information.)

| Site | Object Name |
|---|---|
| Brigham & Women's | urn:eda:rmi:NIH.NCI.EDRN.BRIGHAM.PRODUCT_SERVER |
| Creighton U. | urn:eda:rmi:NIH.NCI.EDRN.CREIGHTON.PRODUCT_SERVER |

| Site | Object Name |
|---|---|
| GLNE | urn:eda:rmi:NIH.NCI.EDRN.DARTMOUTH.PRODUCT_SERVER |
| H.Lee Moffitt | urn:eda:rmi:NIH.NCI.EDRN.MOFFITT.PRODUCT_SERVER |
| M.D. Anderson | urn:eda:rmi:NIH.NCI.EDRN.MDANDERSON.PRODUCT_SERVER |
| U. Pittsburgh | urn:eda:rmi:NIH.NCI.EDRN.PITTSBURGH.PRODUCT_SERVER |
| UCHSC | urn:eda:rmi:NIH.NCI.EDRN.COLORADO.PRODUCT_SERVER |
| UTHSCSA | urn:eda:rmi:NIH.NCI.EDRN.SANANTONIO.PRODUCT_SERVER |

Note the *object name* in the right column of the node you want to query. You'll need it to make an HTTP transaction.

### Query Expressions

All of the product servers deployed throughout ERNE expect a query expression that describes the kinds of specimens in which you're interested, and what details about those specimens to return. You form the query using a boolean expression that names the CDEs to constrain on and the CDEs to return. Here's an example:

```
SPECIMEN_COLLECTED_CODE = 3
  OR SPECIMEN_COLLECTED_CODE = 4
  AND RETURN = BASELINE_CANCER-ICD9-CODE
  AND RETURN = SPECIMEN_AMOUNT_REMAINING_VALUE
```

This query will match all blood (code 3) and bone marrow (code 4) specimens, and will return two attributes about each one, the ICD9 code and how much is remaining.

In general, you can form any complex boolean query using the following relational operators with each CDE:

- `=` or `EQ` for "equals"
- `!=` or `NE` for "not equals"
- `<` or `LT` for "less than"
- `<=` or `LE` for "less than or equal to"
- `>` or `GT` for "greater than"
- `>=` or `GE` for "greater than or equal to"
- `LIKE` for a string match with `%` acting as a wildcard
- `NOTLIKE` for an inverse string match with `%` acting as a wildcard

You use the CDEs on the left hand side of a relational operator to constrain what to retrieve. You use them on the right hand side with the special pseudo-element `RETURN` to select what attributes to retrieve.

You can then link constraints together with these logical operators:

- `AND` or `&` for a logical intersection
- `OR` or `|` for a logical union
- `NOT` or `!` for a ligical negation
- Parentheses `(` and `)` for grouping

Note that it doesn't matter what logical operators you use for attributes to `RETURN`. All `RETURN` expressions are processed separately and are used to select attributes.

For a list of the current CDEs, see the EDRN Secure Website .

## Retrieving Products

Once you know the *object name* and have formed the *query expression*, you're ready to get specimen data products. You get products by accessing a web service at the URL:

```
https://ginger.fhcrc.org/prod
```

You pass request parameters through that URL that specify the object name of the server to query and the query expression. Here's an example:

```
https://ginger.fhcrc.org/prod?
object=urn:eda:rmi:NIH.NCI.EDRN.SANANTONIO.PRODUCT_SERVER&
keywordQuery=SPECIMEN_COLLECTED_CODE+%3D+3+AND+
RETURN+%3D+SPECIMEN_AMOUNT_REMAINING_VALUE
```

We've broken this URL across multiple lines for readability, but if you look closely, you'll see we're querying UTHSCSA for how much is left of all blood (code 3) specimens.

This interface accepts both HTTP GET and HTTP POST style requests, as detailed in the HTTP specification . GET requests are generally easier; they encode the parameters to a request right in the URL string. POST requests transmit the parameters of the request separately. Most HTTP clients, including web browsers, as well as HTTP APIs in programming languages, support both. For this document, we'll be using GET requests. The choice of which to use is up to you.

### Request Parameters

You pass in two request parameters to the `https://ginger.fhcrc.org/prod` web service:

`object`

Contains the *object name* of the product server to query.

`keywordQuery`

Contains the *query expression*.

In the example query above, the `object` parameter was set to the object name of the UTHSCSA product server, `urn:eda:rmi:NIH.NCI.EDRN.SANANTONIO.PRODUCT_SERVER`. The `keywordQuery` parameter was set to the query expression `SPECIMEN_COLLECTED_CODE = 3 AND RETURN = SPECIMEN_AMOUNT_REMAINING_VALUE`.

Request parameters must be properly encoded by the rules of Uniform Resource Identifiers whether they're part of the URL in an HTTP GET or transmitted separately in an HTTP POST. In general, this means that characters like colons in object names as well as spaces and equals signs in query expressions need to be properly escaped. Most browsers and HTTP libraries will happily relax these rules, though, especially in cases where there is no ambiguity. For ERNE queries, this means changing your spaces to `+`s and your equals signs to `%3D`s.

In the example query above, we got away without escaping the colons in the object name. A more conforming URL might look like the following:

```
https://ginger.fhcrc.org/prod?
object=urn%3Aeda%3Armi%3ANIH.NCI.EDRN.SANANTONIO.PRODUCT_SERVER&
keywordQuery=SPECIMEN_COLLECTED_CODE+%3D+3+AND+
RETURN+%3D+SPECIMEN_AMOUNT_REMAINING_VALUE
```

While HTTP APIs may relax the rules about what can go in, they often provide utility functions that will perform appropriate escapes for you. Consult your API documentation for more details.

### Responses

Responses back from the web service at `https://ginger.fhcrc.org/prod` contain a data product which is a table with the results of your specimen search. The result is plain text and uses tabs to separate each column of data (corresponding to the `RETURN` CDEs) and dollar-signs `$` to separate each row.

Here's an example: we'll query Creighton University for all specimens taken from patients whose age at diagnosis of cancer was 90 years or older, and return the participant ID and ICD9 code for each matching specimen. The web service query looks like this:

```
https://ginger.fhcrc.org/prod?
object=urn:eda:rmi:NIH.NCI.EDRN.CREIGHTON.PRODUCT_SERVER&
keywordQuery=BASELINE_CANCER-AGE-DIAGNOSIS_VALUE+%3E+90+AND+
RETURN+%3D+STUDY_PARTICIPANT_ID+AND+RETURN+%3D+BASELINE_CANCER-ICD9-CODE
```

The results of this query (at the time of this writing) are:

```
95044251    157$95044251   157$95044251   157$95044251   157$
95022117    153$95022117   153$95022117   153$95022117   153$
95022117    153$95022117   153$95022117   153$95022117   153$
95022117    153$95022117   153$95022117   153$95022117   153$
95041299    182$95041299   182$95017265   157$95017265   157$
```

```
95017265    157$
```

(The spaces above are tabs; we've broken this result across multiple lines for readability.) Reading left-to-right, you can see that the first specimen is from patient 95044251, and the ICD9 code is 157. This repeats three more times, meaning there are four such specimens available. Then there's a specimen for patient 95022117 with ICD9 code 153 (11 such specimens). Then patient 95041229 with ICD9 code 182. Then patient 95017265 with ICD9 code 157 (3 specimens).

You can process these results in any way you wish. For example, you might develop a VB macro for Excel that inserts results into a spreadsheet, or develop a Java program that formats them into an invoice for printing, and so forth.

### Querying from Applications

As we've mentioned, you're not limited to retrieving product data solely with a web browser. Most programming languages include either direct or add-on support for doing HTTP. In this way, you can develop analysis tools that retrieve ERNE specimen data products from the web service. To demonstrate this, here is a complete Python script that retrieves specimen data and formats it as an XML document.

```
#!/usr/bin/python

import string
import urllib2
import xml.dom.minidom

impl = xml.dom.minidom.getDOMImplementation()
doc = impl.createDocument(None, 'specimens', None)
root = doc.documentElement
product = urllib2.urlopen('https://ginger.fhcrc.org/prod?

object=urn:eda:rmi:NIH.NCI.EDRN.CREIGHTON.PRODUCT_SERVER&

keywordQuery=BASELINE_CANCER-AGE-DIAGNOSIS_VALUE+%3E+90+AND+

RETURN+%3D+STUDY_PARTICIPANT_ID+AND+RETURN+%3D+BASELINE_CANCER-ICD9-CODE')
for row in product.read().split('$'):
        columns = row.split('\t')
        if len(columns) == 2:
                patient = columns[0]
                icd9 = columns[1]
                specimen = doc.createElement('specimen')
                specimen.setAttribute('patientID', patient)
                specimen.setAttribute('icd9', icd9)
                root.appendChild(specimen)
print doc.toprettyxml('  ')
```

When run, the following is output:

```
<?xml version="1.0" ?>
<specimens>
  <specimen icd9="157" patientID="95044251"/>
  <specimen icd9="157" patientID="95044251"/>
  <specimen icd9="157" patientID="95044251"/>
  <specimen icd9="157" patientID="95044251"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="153" patientID="95022117"/>
  <specimen icd9="182" patientID="95041299"/>
  <specimen icd9="182" patientID="95041299"/>
  <specimen icd9="157" patientID="95017265"/>
  <specimen icd9="157" patientID="95017265"/>
  <specimen icd9="157" patientID="95017265"/>
</specimens>
```

# 1.3 Using Profiles

## Using ERNE Profiles

The Early Detection Research Network can use the OODT framework to distribute *profiles* of ERNE resources. Profiles are metadata descriptions of resources. ERNE currently runs no profile servers, however, several are planned. You can use these future profile servers to retrieve metadata descriptions of resources, and discover the locations of resources.

To successfully use profiles with ERNE, you'll need to understand:

- The information captured by profiles
- The representation of profiles
- Queries you can make to ERNE profile servers

# 1.3.1 Information Captured in a Profile

## Information Captured by Profiles

A profile serves as a generic template for describing the characteristics of a resource. Within ERNE, profiles will exist to describe various resources that can include specimen records, documents, web pages, and other items.

## Information and Organization

Profiles capture three kinds of information:

- **Resource Attributes**

  Resource attributes are metadata about the resource's *inception*. These attributes include the creator of the resource, in what language it exists, when it was created, and so forth. These attributes are based on the work of the Dublin Core Metadata Initiative .

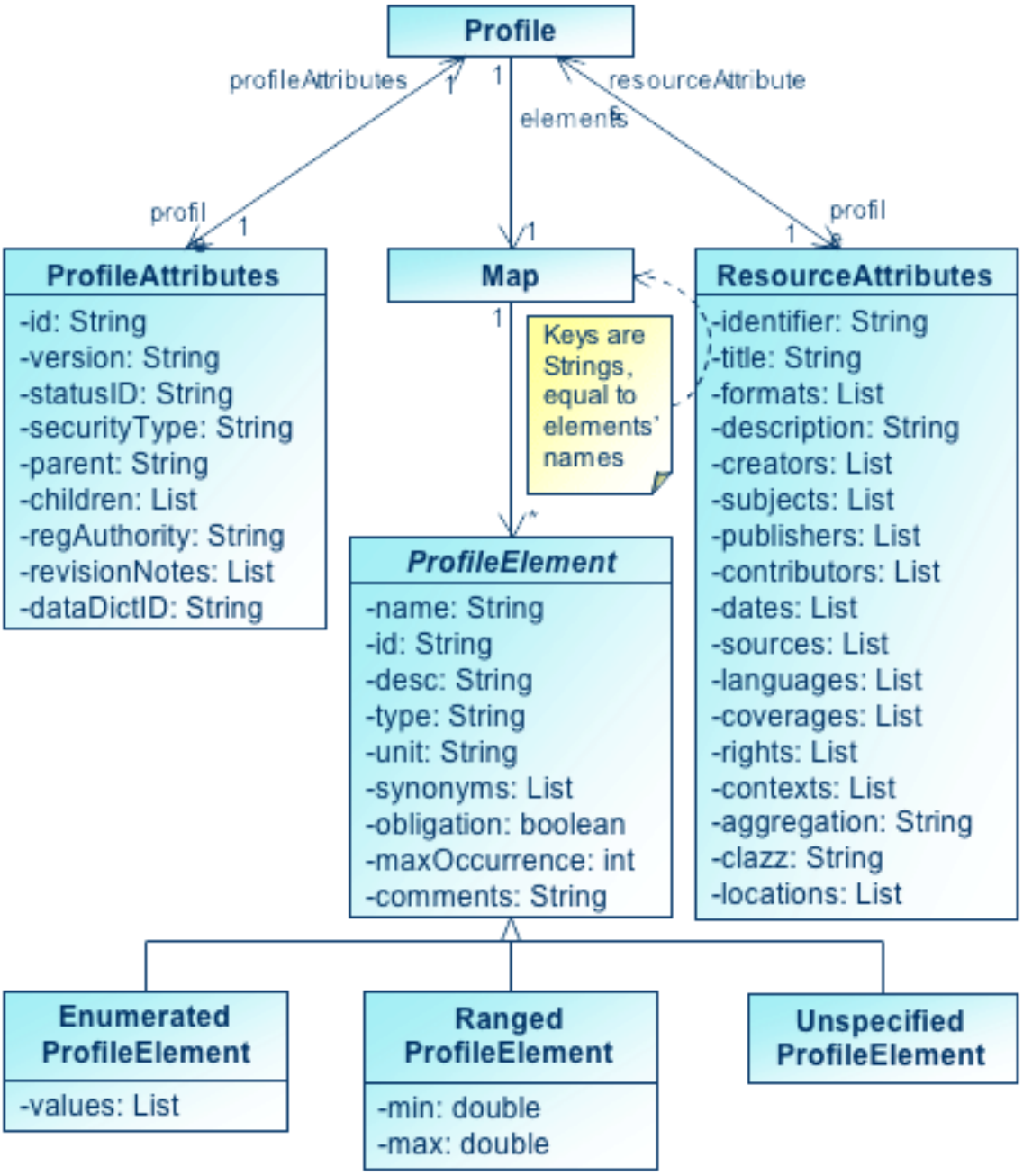
- **Profile Elements**

  Profile elements are metadata about the resource's *composition*. These tell you about the morphology of the resource, such as data types captured within in, minimum and maximum values, synonymous elements, and so forth. These attributes are based on ISO/IEC 11179 standards .

- **Profile Attributes**

  Profile attributes are metadata about the *profile itself*, such as who made it, whether it's classified, revision notes, and so forth. It also has a unique identifying Object Identifier (OID) .

The following class diagram shows the relationship between the different parts of a profile:




While this diagram shows the Java field names and Java classes, the relationship applies to profiles whether they exist as Java objects, as RDF documents, or as XML documents in the profile vocabulary.

## Inception Metadata

Profiles, whether expressed in RDF or in their own XML vocabulary, have a section for capturing information about the resource's inception. This includes information about when the resource was created,

who created it, in what language it exists, and so forth. Profiles use the element set recommended by the Dublin Core Metadata Initiative (DCMI) set in order to describe the inception of a resource, with some extensions.

Collectively, these metadata are called the *resource attributes* or `resAttributes` of the profile. Every profile has one and only one set of `resAttributes`. The metadata elements within the `resAttributes` are defined in this section.

#### Identifier

As defined by the DCMI, the `Identifier` of a resource is some unambiguous way to identify the resource. In the profile implementation, one and only one `Identifier` is *required.*

#### Title

The `Title` names the resource, and is the name by which the resource is formally known. The `Title` is optional; if present, it may occur only once in a profile.

#### Format

The `Format` indicates the manifestation of the resource. The MIME type is usually recorded here.

#### Description

The `Description` element contains a free text account of the content of the resource. It's optional in a profile; if present, it may occur only once.

#### Creator

Zero or more `Creator`s may be specified in a profile. `Creator`s contain the name of people or organizations that created the resource.

#### Subject

Zero or more `Subject`s in a profile act as keywords. The purpose of the `Subject` elements is to contain a keywords that describe the resource, usually selected from a controlled vocabulary.

#### Publisher

Any number of `Publisher` elements may appear in a profile. They contain the organization responsible for making the resource available.

#### Contributor

A `Contributor` is a person or organization providing auxilliary work towards the resource's creation. Any number of `Contributor`s may be listed in a profile.

#### Date

`Date` elements indicate the times in history when the resource was created. Any number of `Date`s may be included in a profile.

#### Type

The `Type` element indicates the nature of the content of the resource, such as "fiction" for a work of fiction or "image" for a dataset rendered graphically.

#### Source

When a resource is derived others, the `Source` element indicates the `Identifier`s of the referenced resources.

#### Language

For resources that contain natural language content, the `Language` element indicates the languages in use.

#### Relation

When a resource is related to others, a profile can specify the `Identifier`s of the related resources using zero or more `Relation` elements.

#### Coverage

For resources that cover a space or time or jurisdiction, use the `Coverage` element to indicate such coverage. This element may be listed any number of times in a profile, and its content usually comes from a controlled vocabulary.

#### Rights

Copyright, ownership, redistribution, use, and other legal issues may exist for a resource. When that happens, the `Rights` element lists the rights management information.

*Note:* The official name of element for is plural `Rights`; this is inconsistent with the other metadata elements, but is consistent with the DCMI.

#### resContext

The `resContext` element identifies the application environment or discipline within which the resource originates and is derived from a taxonomy of scientific disciplines. This element is required in a profile and may occur multiple times.

As an example, a `resContext` of `EDRN.Study.SELDI` tells that the resource is associated with the SELDI Study under the EDRN.

#### resAggregation

The `resAggregation` element indicates the aggregative structure of the resource. It tells you what you'll get if you retrieve the resource: a granule, a dataset, or a collection of datasets. The legal values of this optional elements are:

- `granule`, meaning the resource is a single product
- `dataSet`, meaning the resource is a set of products
- `dataSetCollection`, meaning the resource is collection of datasets

The `resAggregation` element is optional; however, if specified, it may appear in a profile only once.

#### resClass

The `resClass` element identifies the kind of the resource within a taxonomy of resource types. It's a *required* element that is used by the OODT Framework to determine how to treat the profile as well as the resource named by the profile.

For example, a `resClass` of `system.productServer` indicates that the resource is an OODT product server. A query that matches this profile means that if the same query were given to the identified product server, it would yield a result. A `resClass` of `system.profileServer` means the resource is a profile server. That means that while the current profile server may or may not provide a matching profile, another profile server might, forming an implicit digraph of profile servers. Other valid `resClass` values include `data.granule`, `data.dataSet`, and `application.interface`.

#### resLocation

Zero or more `resLocation` elements may appear in a profile. They tell where the resource is located, easily the most important part of the profile. Because this element may appear several times, all locations should be considered valid; the application may pick the one that's most convenient. The `resLocation` may also appear zero times. This means that the profile indicates solely that the resource exists, but where is unknown.

The interpretation of the resLocation is as a URI. For example, a `resClass` of `system.productServer` or `system.profileServer` means that the `resLocation` indicates an URN to a software object name. Querying that object will yield either the desired result (for product servers) or more matching profiles (for

profile servers). For a resClass of `data.granule` or `data.dataSet`, the `resLocation` is an URL to the granule or dataset.

## Composition Metadata

The most interesting part of a profile is in the metadata that describes the composition of the resource that the profile profiles. The composition metadata is what enables a profile server to tell if a particular resource can answer a query.

The composition metadata is based on the data element description standards in ISO/IEC standard 11179. They are the *profile elements* or `profElement`s of a profile. Every profile may have zero or more `profElement`s, the components of which are discussed in this section.

#### elemId

The `elemId` is an optional universally unique identifier applied to the element.

#### elemName

The `elemName` is the *required* name of the profile element. It serves as the title role of one of the components of the resource.

#### elemDesc

The `elemDesc` is the description of the profile element. Although the title may often be enough to identify the purpose of the profile element, the description should be used to provide any further, free-text information that may be of importance to analysts and profile administrators.

#### elemType

The `elemType` indicates the type of data represented in the profile element, synonymous to the ISO/IEC 11179 `Datatype` attribute. The permissible values are:

- `boolean`
- `character`
- `date_time`
- `enumerated`
- `integer`
- `ordinal`
- `rational`
- `scaled`

- `real`
- `complex`
- `state`
- `void`

This element is optional within a profile element. When it's not present, the profile element merely indicates that the resource's content possesses the attribute, but more is not known.

#### elemUnit

The `elemUnit` indicates the units associated with the values of the data element. This element is synonymous to the ISO/IEC 11179 attribute `unit.of.quantity`. Values for this optional element should be selected from standardized tables of units.

#### elemEnumFlag, elemValue, elemMinValue, and elemMaxValue

The `elemEnumFlag` tells how possible values of the profile element are specified. It works with the `elemValue`, `elemMinValue`, and `elemMaxValue` elements:

- If the `elemEnumFlag`'s value is `T` and one or more `elemValue`s appear, then the values listed are the valid values of the element.
- If the value is `F`, then a closed range of values bounded by the profile's `elemMinValue` and `elemMaxValue` elements indicates the valid values.
- If the value is `T` but no `elemValue`s appear, then it means that any value is a valid value for the resource.

#### elemSynonym

Often, a characteristic of a resource will go by several names, especially between scientific disciplines. What one person may call *latitude*, another may call *x coordinate*, for example.

The `elemSynonym` provides a way to indicate synonyms. Zero or more `elemSynonym`s may appear in a profile element. The values of this element are names from data dictionaries other than the discipline data dictionary hosting the profile.

#### elemComment

The `elemComment` field provides a remark concerning the application of the data element. This element is synonymous to the ISO/IEC 11179 attribute `Comment`, and is optional within a profile element.

### Metadata about the Profile

For a profile server to manage a set of profiles, it's necessary to have metadata contained within the profile that describes the profile itself. This metadata, collectively called the profile attributes, or `profAttributes`,

serves that purpose.

Most of the elements within the `profAttributes` are optional. This sections describes each of them.

#### profId

The `profId` serves to give a unique identifier to the profile. It may be expressed as a URI, and often as an URN, or as an OID.

#### profVersion

The `profVersion` identifies the version number of the profile.

#### profType

The `profType` identifies the type of the profile. The type that typically appears here is `profile`, meaning the profile is a profile (obviously).

Another type that can be here is `dataDict`, which indicates that the profile doesn't describe a resource, but instead is a data dictionary for other profiles. Such a profile's composition elements name the expected profile elements and ranges of valid valuese that will appear in other profiles. The `profDataDictId` element identifies the profile serving as its data dictionary.

#### profStatusId

The `profStatusId` identifies the state of the profile. Profiles may be either `active` or `inactive`. An inactive profile is likely maintained for historical or exemplary reasons but is otherwise not currently used for searches or resource descriptions.

#### profSecurityType

The `profSecurityType` identifies whether the information contained in the profile may be of a sensitive nature. Any string is valid here.

#### profParentId

The `profParentId` optionally identifies the URI of the parent of this profile. Profiles may be arranged hierarchically in a singly rooted tree in a forest.

#### profChildId

The `profChildId` identifies zero or more children (by duplicating the element) of this profile.

#### profRegAuthority

The `profRegAuthority` names the registration authority responsible for authoring and maintaining the profile.

#### profRevisionNote

The `profRevisionNote` appears zero or more times in the profile to describe changes made to it over time. The notes are free form text, and each element is ordered from newest to oldest note.

#### profDataDictId

The `profDataDictId` identifies the profile providing a data dictionary to the current profile.

# 1.3.2 Representation of Profiles

## Representation of ERNE Profiles

As you've surmised by now, Profiles exist as Java objects within ERNE. However, they can also be represented as RDF documents or XML documents. When you query the ERNE web service, you'll receive them as XML documents, from which you can then re-construct Java objects, or manipulate them directly, if you're not using Java.

## XML Profiles

Profiles can be represented as XML documents that conform to the OODT Profile Document Type Definition (DTD). The Formal Public Identifier of the OODT Profile DTD is `-//JPL//DTD Profile 1.0//EN`. The normative System Identifier is http://oodt.jpl.nasa.gov/grid-profile/dtd/prof.dtd.

Although you should refer to the normative System Identifier for the latest reference version, see the following for the DTD:

```
<!ELEMENT profiles
  (profile*)>

<!ELEMENT profile
  (profAttributes,
   resAttributes,
   profElement*)>

    <!ELEMENT profAttributes
      (profId, profVersion?, profType,
       profStatusId, profSecurityType?, profParentId?, profChildId*,
       profRegAuthority?, profRevisionNote*)>

    <!ELEMENT resAttributes
      (Identifier, Title?, Format*, Description?, Creator*, Subject*,
       Publisher*, Contributor*, Date*, Type*, Source*,
       Language*, Relation*, Coverage*, Rights*,
       resContext+, resAggregation?, resClass, resLocation*)>

    <!ELEMENT profElement
      (elemId?, elemName, elemDesc?, elemType?, elemUnit?,
       elemEnumFlag, (elemValue* | (elemMinValue, elemMaxValue)),
       elemSynonym*,
       elemObligation?, elemMaxOccurrence?, elemComment?)>

    <!ELEMENT profId (#PCDATA)>
```

```
<!ELEMENT profVersion (#PCDATA)>
<!ELEMENT profType (#PCDATA)>
<!ELEMENT profParentId (#PCDATA)>
<!ELEMENT profChildId (#PCDATA)>
<!ELEMENT profStatusId (#PCDATA)>
<!ELEMENT profSecurityType (#PCDATA)>
<!ELEMENT profRegAuthority (#PCDATA)>
<!ELEMENT profRevisionNote (#PCDATA)>

<!ELEMENT Identifier (#PCDATA)>
<!ELEMENT Title (#PCDATA)>
<!ELEMENT Format (#PCDATA)>
<!ELEMENT Description (#PCDATA)>
<!ELEMENT Creator (#PCDATA)>
<!ELEMENT Subject (#PCDATA)>
<!ELEMENT Publisher (#PCDATA)>
<!ELEMENT Contributor (#PCDATA)>
<!ELEMENT Date (#PCDATA)>
<!ELEMENT Type (#PCDATA)>
<!ELEMENT Source (#PCDATA)>
<!ELEMENT Language (#PCDATA)>
<!ELEMENT Relation (#PCDATA)>
<!ELEMENT Coverage (#PCDATA)>
<!ELEMENT Rights (#PCDATA)>
<!ELEMENT resContext (#PCDATA)>
<!ELEMENT resAggregation (#PCDATA)>
<!ELEMENT resClass (#PCDATA)>
<!ELEMENT resLocation (#PCDATA)>

<!ELEMENT elemId (#PCDATA)>
<!ELEMENT elemName (#PCDATA)>
<!ELEMENT elemType (#PCDATA)>
<!ELEMENT elemEnumFlag (#PCDATA)>
<!ELEMENT elemDesc (#PCDATA)>
<!ELEMENT elemSynonym (#PCDATA)>
<!ELEMENT elemUnit (#PCDATA)>
<!ELEMENT elemValue (#PCDATA)>
<!ELEMENT elemMinValue (#PCDATA)>
<!ELEMENT elemMaxValue (#PCDATA)>
<!ELEMENT elemObligation (#PCDATA)>
<!ELEMENT elemMaxOccurrence (#PCDATA)>
<!ELEMENT elemComment (#PCDATA)>
```

#### Collections of Profiles

As you can see from the above,an XML element, `profiles` is a container element to hold zero or more profiles. Responses from ERNE servers always start with this element.

## Java Representation of Profiles

The OODT source code includes a class `jpl.eda.profile.Profile` for object represenation of a

profile. You can construct a `Profile` object from an XML document or create a blank one to populate with metadata later. See the Profile Javadocs for more information.

#### Accessing Profile Metadata

To access the metadata of a profile, call the methods to retrieve the profile attributes, the resource attributes, or the profile elements.

#### Accessing the Profile Attributes

You retrieve the profile attributes by calling `getProfileAttributes` on a `Profile`. This returns an `ProfileAttributes` object which provides methods to get and set the various attributes. Setting a value sets it for the `Profile` to which the `ProfileAttributes` belongs.

A value of `null` for an optional attribute means the value isn't set.

#### Accessing the Resource Attributes

You retrieve the resource attributes similarly as for profile attributes, calling `getResourceAttributes` to yield a `ResourceAttributes` object.

The `ResourceAttributes` has methods to get and set various attributes. Note that many of the attributes are multi-valued. For example, the resource profiled will likely cover several subjects. In this case, the "get" method, `getSubjects`, returns a `java.util.List` of `String`s. There is no set method. Instead, you just manipulate the list to add and remove subjects.

*Note:* All of the "get" functions that return `List`s return lists of `String`s, except for `getDates`, which returns a list of `java.util.Date`s.

For other attributes which are singly valued, there is both a set and get method. For optional values, a value of `null` means the attribute isn't set.

#### Accessing the Profile Elements

The profile stores its profile elements in a `java.util.Map`, mapping the name of the profile element (as a `String`) to an object of class `ProfileElement`. To access this map, call the method `getProfileElements` on a `Profile`. Because this method returns a reference to the `Profile`'s map, any updates to the map affect the profile immediately.

*Caution:* Never store anything but `String`s as keys and `ProfileElement`s (or objects of its subclasses) as values in a profile element map. The software will not operate correctly if any other kind of object is stored.

#### Common Attributes of Profile Elements

The class `jpl.eda.profile.ProfileElement` contains the common parts of every profile element,

such as its required name, its optional description, and so forth. Use the value `null` for any optional attribute that's unset.

The `ProfileElement` stores its synonyms as a `java.util.List` of `String`s. Manipulate the list directly to add or remove synonyms.

The `ProfileElement` class is abstract. To create new profile elements for a profile, you need to create objects of one of the concrete `ProfileElement` subclasses:

- `EnumeratedProfileElement`
- `RangedProfileElement`
- `UnspecifiedProfileElement`

The following sections detail each kind of profile element.

#### Elements with Enumerated Values

For profile elements that maintain a specified list of valid values, use the `EnumeratedProfileElement` class. Objects of this class maintian a `java.util.List` of values. You can pass in a list of values when constrcuting the object, or can call the `getValues` method and manipulate the list directly.

Queries that arrive for an enumerated profile element must match one of the listed elements exactly unless it's a negative (not-equal-to) query. For example, suppose we had an enumerated profile element `filter` with values `infrared`, `visible`, and `ultraviolet`. A query asking for a filter equal to infrared should match, as well as a query asking for a filter greater than or equal to infrared. A query asking for a filter not equal to infrared shouldn't match, while a query asking for a filter not equal to x-ray should match.

#### Elements with a Range of Values

You can represent profile elements that have a range of valid values with the `RangedProfileElement` class. Construct this class with the minimum and maximum values, which must be numeric.

Use the `getMinValue` to get the minimum value and `getMaxValue` to get the maximum value.

Querying a profile that uses a range of values considers the range as inclusive. For example, suppose the ranged profile element `temperature` has a minimum value of 32 and a maximum value of 212. A query that requests temperatures less than 32 shouldn't match, but less than or equal to 32 should match.

#### Elements with no Specified Values

For profile elements that you always want to match a query without explicitly listing each valid value or a range of legal numeric values you can use the class `UnspecifiedProfileElement` . This class identifies an element with no range or list of valid values.

Queries that arrive at such an element will *always* match, even if they're negative (not-equal-to) queries.

# 1.3.3 Querying Profile Servers

## Querying ERNE Profile Servers

To retrieve a profile of a resource, you send a query to a ERNE web service. It, in turn, passes it to your selected profile server, which queries its data stores and synthesizes profile objects. Finally, the web service returns those matching profile objects in an XML document. You can then examine and manipulate this XML document directly or construct Java profile objects out of it and manipulate them.

## The Query Web Service

Retrieving profiles via HTTP uses a web service available at the ERNE single point of entry website:

```
https://ginger.fhcrc.org/q
```

Here the "q" means "query," and it enables you to pass queries into the OODT framework serving ERNE. Profile queries will respond with XML documents representing a set of matching profiles.

### Making Requests

To retrieve profiles using this web service, you need to know two things:

1. The *object name* of the profile server to which you'd like the query directed.
2. The *query expression* that selects what profiles to retrieve.

The object name selects a single profile server from the various profile servers available. The query expression selects profiles. Different profile servers accept different expressions and each serve different kinds of metadata profiles.

Since ERNE does not yet define any profile servers, none are currently available for querying.

The web service at `https://ginger.fhcrc.org/q` supports both HTTP GET and HTTP POST style requests to retrieve profiles, as detailed in the HTTP specification . GET requests are generally easier; they encode the parameters to a request right in the URL string. POST requests transmit the parameters of the request separately. Most HTTP clients, including web browsers, as well as HTTP APIs in programming languages, support both. For this document, we'll be using GET requests. The choice of which to use is up to you.

When calling the web service at `https://ginger.fhcrc.org/q`, you need to pass in three parameters:

type

The type tells what kind of query you're making. This should always be set to `profile`.

object

This tells what profile server to query.

keywordQuery

This is the query expression to pass to the profile server

As with all most web services, you'll need to encode your parameters according to the rules in Uniform Resource Identifiers whether they're part of the URL in an HTTP GET or transmitted separately in an HTTP POST. In general, this means that characters like colons in object names as well as spaces and equals signs in query expressions need to be properly escaped. Most browsers and HTTP libraries will happily relax these rules, though, especially in cases where there is no ambiguity. For ERNE profile queries, this means changing your spaces to `+`s and any equals signs to `%3D`s.

### From XML to Java

If you're not into XML, you might find it more convenient to work with Java objects that represent profiles and their attributes. Creating Java objects out of the XML document returned by the `https://ginger.fhcrc.org/q` web service is really quite simple. First, make the URL to query the service just as above. Open the URL and parse the response document into a Document Object Model (DOM) tree. Then call the createProfiles method on the root element of the DOM tree.

Let's look at an example. Note that since ERNE has no profile servers yet, this is a hypothetical example.

```
import java.util.Iterator;
import java.util.List;
import java.util.Map;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import jpl.eda.profile.Profile;
import jpl.eda.profile.ProfileAttributes;
import jpl.eda.profile.ProfileElement;
import jpl.eda.profile.ResourceAttributes;
import org.w3c.dom.Document;

public class UseProfiles {
  public static void main(String[] argv) throws Throwable {
    String url = "https://ginger.fhcrc.org/q?"
      + "type=profile&object=urn:eda:rmi:EDRN.Profile"
      + "&keywordQuery=CELL_COUNT+=+%3D+200";
    DocumentBuilderFactory fac = DocumentBuilderFactory.newInstance();
    DocumentBuilder builder = fac.newDocumentBuilder();
    Document doc = builder.parse(url);
    List profiles = Profile.createProfiles(doc.getDocumentElement());
    for (Iterator i = profiles.iterator(); i.hasNext();) {
      Profile p = (Profile) i.next();
      ProfileAttributes a = p.getProfileAttributes();
      ResourceAttributes r = p.getResourceAttributes();
      System.out.println("PROFILE " + a.getID() + ":");
```

```
      System.out.println("\tResource title: " + r.getTitle());
      List locs = r.getResLocations();
      System.out.println("\tLocations (" + locs.size() + "):");
      for (Iterator j = locs.iterator(); j.hasNext();)
        System.out.println("\t\t" + j.next());
      Map elems = p.getProfileElements();
      System.out.println("\tElements (" + elems.size() + "):");
      for (Iterator j = elems.values().iterator(); j.hasNext();) {
        ProfileElement e = (ProfileElement) j.next();
        System.out.println("\t\tName: " + e.getName());
        System.out.println("\t\tLegal values: " + e.getValues());
        System.out.println("\t\tMin value: " + e.getMinValue());
        System.out.println("\t\tMax value: " + e.getMaxValue());
      }
    }
  }
}
```

This program sends a query to the web service and parses the response into an in-memory DOM tree. It then creates a `java.util.List` of `Profile` objects out of that tree, and iterates through the list, printing some details about each `Profile`.

To compile and run this program, you'll need two other components:

- Profile Service , which contains definitions for classes such as `Profile`, `ProfileAttributes`, and so forth.
- OODT Common Components , which contains basic utilities used by all other OODT components.

Since Java tools and integrated development environments vary, consult your own documentation on how to make your system aware of these components. If you're using the Java command-line tools, then download each of the binary distributions of the above components and copy the jar file from each into a directory, say `lib`. Then put `UseProfiles.java` into a directory called `src`. You can then compile `UseProfiles.java` into a directory called `classes` and run the class:

```
% ls -l lib
total 344
-rw-r--r--  1 kelly  kelly  144169 12 Mar 07:41 edm-commons-2.2.4.jar
-rw-r--r--  1 kelly  kelly  201451 12 Mar 07:41 grid-profile-3.0.2.ja
% ls -l src
total 4
-rw-r--r--  1 kelly  kelly  1950 12 Mar 07:42 UseProfiles.java
% mkdir classes
% javac -extdirs lib -d classes src/UseProfiles.java
% ls -l classes
total 4
-rw-r--r--  1 kelly  kelly  2810 12 Mar 07:42 UseProfiles.class
% java -Djava.ext.dirs=lib -classpath classes UseProfiles
PROFILE 1:
         Identifier: ...
```

Again, note that this is a hypothetical example of how the system may work, as there are no profile servers defined yet for ERNE.